The ToxCast Analysis Pipeline: An R Package for Processing and Modeling Chemical Screening Data

Dayne Lewis Filer

December 1, 2014

Contents

Introduction **2**
- Overview . . . 2
- Package Settings . . . 2
- Assay Structure . . . 4

Multiple Concentration Screening **5**
- First Steps . . . 5
- Level 1 . . . 7
- Level 2 . . . 10
- Level 3 . . . 13
- Level 4 . . . 15
- Level 5 . . . 19
- Level 6 . . . 23

Single Concentration Screening **27**

A Field Explaination/Database Stucture **28**

B Level 0 Pre-Processing **39**

C Burst Z-Score Calculation **41**

D Plans for V1.0 Release **43**

DRAFT VERSION

Introduction

Overview

This package was developed to process the high-throughput and high-content screening data generated by the US EPA ToxCast^{TM} program.[1] The ToxCast program is screening thousands of chemicals with hundreds of assays coming from numerous and diverse biochemical and cell-based technology platforms. The diverse data received from in heterogeneous formats from numerous vendors is transformed to a standard computible format and loaded into the `tcpl` databases by vendor-specific R scripts. Once data is loaded into the databases, the ToxCast program utilizes the generalized processing functions provided in this package to process, normalize, model, qualify, flag, inspect, and visualize the data.

The ToxCast program includes two screening paradigms: single concentration screening and multiple concentration screening. Single concentration screening consists of testing chemicals at one concentration – often for the purpose of identifying potential actives to test at multiple concentrations. Multiple concentration screening consists of testing chemicals across a concentration range, such that the modeled activity can give an estimate of potency, efficacy, etc.

Prior to the pipeline processing provided in this package, all the data must go through pre-processing (level 0). Level 0 pre-processing utilizes dataset-specific R scripts to process heterogeneous data into a uniform format and load the uniform data into the `tcpl` databases. Level 0 pre-processing is outside the scope of this package, but can be done on virtually any high-throughput or high-content screening efforts, provided the resulting data includes the minimum required information.

In addition to storing the data, the `tcpl` databases store every processing/analysis decision at the assay component or assay endpoint level to facilitate transparency and reproducibility. For the illustrative purposes in this vignette we have included SQLite versions of the `tcpl` databases containing dummy data. Due to differences in database capabilities, not all functionality of the package will work with the SQLite version. To best utilize the package the user should work with MySQL databases and the `RMySQL` package. The MySQL versions of the `tcpl` databases containing all the publicly available ToxCast data are available for download at: `<http://epa.gov/ncct/toxcast/data.html>`.

Package Settings

First we will load the package (you will also want to load the `data.table` package), and look at the default settings for the example databases.

[1]`<http://www.epa.gov/ncct/toxcast/>`

R Input

```
> library(data.table)
> library(tcpl)
```

When you load the package in the R console, you will see a message like the following:

R Output

```
tcpl loaded with the following settings:
  TCPL_DATA:  /usr/local/lib64/R/library/tcpl/sql/xmpl.sqlite
  TCPL_CHEM:  /usr/local/lib64/R/library/tcpl/sql/xmpl.sqlite
  TCPL_USER:
  TCPL_HOST:
  TCPL_DRVR:  SQLite
  TCPL_LOG:   /usr/local/lib64/R/library/tcpl
Default settings stored in TCPL.conf. See ?tcplListOpts or
?tcplSetOpts for more information.
```

Here we see `$TCPL_DATA` and `$TCPL_CHEM` point to the the example database in the package directory, and the database driver (`$TCPL_DRVR`) is set to "SQLite". If you choose to download and use the ToxCast database, you will have to configure the settings to reflect your MySQL environment. While the DATA and CHEM databases can be separate, in the ToxCast database currently released the chemical and data tables have been merged into a single database for simplicity. Look at `?tcplSetOpts` for more information. At any time you can check the settings using `tcplListOpts()`. An example of database settings would be:

R Input

```
> tcplSetOpts(drvr = "MySQL",
              user = "root",
              pass = "",
              host = "localhost",
              data = "invitrodb_v1",
              chem = "invitrodb_v1",
              log = system.file(package = "tcpl"))
```

When the package is loaded, the settings are configured by the TCPL.config file located in the package directory. The user can edit the file, such that the package loads with the desired settings every time. The configuration script has to be edited whenever the package is updated or re-installed.

Assay Structure

The definition of an "assay" gets complicated when dealing with high-throughput/high-content data. For the purposes of this package, "assay" is broken into:

assay_source – the vendor/origination of the data

assay – the procedure to generate the component data

assay_component – the raw data readout(s)

assay_component_endpoint – the normalized component data

Each element has a separate table in the `tcpl` databases. In general we refer to an "assay_component_endpoint" as an "assay endpoint". As we move down the hierarchy, each additional layer has a one-to-many relationship with the previous layer. For example, an assay component can have multiple endpoints, but an assay endpoint can only have one assay component.

Assay and assay source do not contain any chemical data, but store annotations to help in the processing and down-stream understanding/analysis of the data. All processing occurs by assay component or assay endpoint, depending on the processing level. For more information about the assay annotation please refer to `<http://www.epa.gov/ncct/toxcast/>`.

Multiple Concentration Screening

This section will cover the `tcpl` process for handling multiple concentration data. The goal of multiple concentration processing is to estimate the activity, potency, efficacy, and other parameters for sample-assay pairs. After the data is loaded into the `tcpl` databases, the multiple concentration processing consists of six levels (Table 1).

Table 1: Summary of the `tcpl` multiple-concentration pipeline.

	Description	Function
Lvl 0	Vendor/dataset-specific pre-processing to transform heterogeneous data to the uniform tcpl format for processing by the `tcpl` package	`N/A`†
Lvl 1	Define the replicate and concentration indices to facilitate all subsequent processing	`tcpl1`
Lvl 2	Apply assay component-specific corrections listed in the `tcpl` databases to the raw data to define the corrected data	`tcpl2`
Lvl 3	Apply assay endpoint-specific normalization listed in the `tcpl` databases to the corrected data to define response	`tcpl3`
Lvl 4	Model the concentration-response data utilizing three objective functions: constant, hill, and gain-loss	`tcpl4`
Lvl 5	Select the winning model, define the response cutoff based on methods in the `tcpl` databases, and determine activity	`tcpl5`
Lvl 6	Flag potential false positive and false negative findings based on methods in the `tcpl` databases	`tcpl6`

†Level 0 processing outside the scope of this package

First Steps

To start exploring the data we will look first in the assay source table to find out what data is available in the `tcpl` databases.

R Input

```
> AS <- tcplLoadAsid()
> AS
```

R Output

```
   asid asnm
1:    1  DEC
```

We can see the example database contains only one assay source, "DEC". What if we want to know more about the assay source? We know from the previous section on assay structure that the assay source information is stored in the assay_source table. The user can use the `tcplListFlds` function to find out what other information is available.

R Input
```
> tcplListFlds(tbl = "assay_source")
```

R Output
```
[1] "asid"                    "assay_source_name"
[3] "assay_source_long_name" "assay_source_desc"
```

R Input
```
> ## We can add the long name and description to the add.fld
> ## variable in the tcplLoadAsid call
> AS <- tcplLoadAsid(add.fld = c("assay_source_long_name",
                                 "assay_source_desc"))
> AS
```

R Output
```
   asid asnm          assay_source_long_name
1:    1  DEC Dayne's Example Biotech Company
                           assay_source_desc
1: Completely fake data for illustration.
```

The convention used in the ToxCast program is to prepend all of the assay component and assay endpoint names with the assay source name (an abbreviation of the assay source long name). We can use the assay source information to see what assay components exist, then look at the level 0 data.

R Input
```
> tcplLoadAcid(fld = "asid", val = 1L)
```

R Output
```
   asid acid           acnm
1:    1    1 DEC_Fake_Assay
```

R Input
```
> l0data <- tcplLoadData(lvl = 0L, fld = "acid", val = 1L)
> dim(l0data)
```

R Output

```
[1] 864  12
```

R Input

```
> head(l0data)
```

R Output

```
   l0id spid cpid acid   apid rowi coli wllt wllq conc
1:    1 DMSO   NA    1 AP0001    2   21    n    1  0.5
2:    2 DMSO   NA    1 AP0001    2   22    n    1  0.5
3:    3 DMSO   NA    1 AP0001    3    2    n    1  0.5
4:    4 DMSO   NA    1 AP0001    4    2    n    1  0.5
5:    5 DMSO   NA    1 AP0001    5    2    n    1  0.5
6:    6 DMSO   NA    1 AP0001    6    2    n    1  0.5
    rval                   srcf
1: 1.001 Some_source_file_1.txt
2: 0.891 Some_source_file_1.txt
3: 0.734 Some_source_file_1.txt
4: 0.760 Some_source_file_1.txt
5: 0.813 Some_source_file_1.txt
6: 0.982 Some_source_file_1.txt
```

In the example dataset we have one assay component ("DEC_Fake_Assay", acid 1) with 864 entries. Appendix A contains a complete description of the field names. The following sections will explain the processing at each level in detail.

Level 1

Level 1 processing defines the replicate and concentration index fields to facilitate downstream processing. Due to cost, availability, physicochemical, and technical constraints screening level efforts utilize numerous experimental designs and test compound (sample) stock concentrations. The resulting data contains inconsistent numbers of concentrations, concentration values, and technical replicates. To enable quick and uniform processing, level 1 processing explicitly defines concentration and replicate indices, giving integer values $1 \dots N$ to increasing concentrations and technical replicates where 1 represents the lowest concentration or first technical replicate.

To assign the replicate and concentration index we assume one of two experimental designs. The first design assumes samples are plated in multiple concentrations on each assay plate, such that the concentration series all falls on a single assay plate. The second design assumes samples are plated in a single concentration on each assay plate, such that the concentration series falls across many assay plates.

For both experimental designs the data is ordered by source file (*srcf*), assay plate ID (*apid*), column index (*coli*), row index (*rowi*), sample ID (*spid*), and concentration (*conc*). Concentration is rounded to three significant figures to correct for potential rounding errors. After ordering the data we create a temporary replicate id, identifying individual concentration series. For test compounds in experimental designs with the concentration series on a single plate and all control compounds, the replicate ID consists of the sample id, well type (*wllt*), source file, assay plate id, chemical plate ID (*cpid*), and concentration. The replicate ID for test compounds in experimental designs with concentration series that span multiple assay plates is defined similarly, but does not include apid.

Once the data is ordered and the temporary replicate ID is defined, we scan the data from top to bottom and increment the replicate index (*repi*) every time a replicate ID is duplicated. Then for each replicate we define the concentration index (*cndx*) by ranking the unique concentrations, with the lowest concentration starting at 1.

We will start by doing the level 1 processing. The multiple concentration data processing is done with the "tcpl#" functions, where the number indicates the level for processing. For level 1 processing we will use `tcpl1`, which takes a single acid. For running multiple ids look at `?tcplRunPipe`. Do the processing and inspect the results:

R Input
```
> ## Do level 1 processing for acid 1
> tcpl1(ac = 1L)
```

R Output
```
Loaded L0 ACID1 (864 rows; 0.01 secs)
Processed L1 ACID1 (864 rows; 0.04 secs)
Completed delete cascade for 1 ids (0.07 secs)
Wrote L1 ACID1 (864 rows; 0.09 secs)
[1] TRUE
```

The processing functions print messages to the console indicating the four steps of the processing. First, the data is loaded, the data is processed, subsequent level is deleted, then the processed data is written to the databse. The function returns a boolean indicating the success of the processing. The "delete cascade" here means all data after level 1 got deleted from the database for the processed acid. The delete cascade ensures data fidelty across the database. Suppose a user decides to rerun level 3 after finishing processing to level 6, but forgets to rerun the subsequent levels. Without the delete cascade, the old data in level 4 through level 6 would not correspond to the new level 3 data in the database. With the delete cascade, the user can look at the database at any time and get a complete picture of the data processing. With the processing complete we can load the level 1 data and check the processing:

R Input

```
> ## Load the level 1 data and look at the assay plate ids
> l1data <- tcplLoadData(lvl = 1L, fld = "acid", val = 1L)
> l1data <- tcplPrepOtpt(l1data)
> setkeyv(l1data, c("repi", "cndx"))
> l1data[chnm == "ChemName3",
         list(chnm, acid, conc, cndx, repi)]
```

R Output

```
         chnm acid  conc cndx repi
 1: ChemName3    1   0.3    1    1
 2: ChemName3    1   1.0    2    1
 3: ChemName3    1   3.0    3    1
 4: ChemName3    1  10.0    4    1
 5: ChemName3    1  30.0    5    1
 6: ChemName3    1 100.0    6    1
 7: ChemName3    1   0.3    1    2
 8: ChemName3    1   1.0    2    2
 9: ChemName3    1   3.0    3    2
10: ChemName3    1  10.0    4    2
11: ChemName3    1  30.0    5    2
12: ChemName3    1 100.0    6    2
13: ChemName3    1   0.3    1    3
14: ChemName3    1   1.0    2    3
15: ChemName3    1   3.0    3    3
16: ChemName3    1  10.0    4    3
17: ChemName3    1  30.0    5    3
18: ChemName3    1 100.0    6    3
```

ChemName3 contains 3 replicates, each with 6 distinct concentrations. The package also contains a tool for visualizing the level 1 - level 3 data at the assay plate level. In Figure 1 we see the results of `tcplPlotPlate`. The row and column indices are printed along the edge of the plate, with the values in each well represented by color. While the plate does not give sample ID information, the letter number codes in the wells indicate the well type and concentration index, respectively. The plate display also shows the wells that did not pass quality with an "X". Plotting plates in subsequent levels will show empty wells where the well did not meet the quality metrics, denoted by the well quality (*wllq*) field in the level 0 table. The title of the display shows the assay component/assay endpoint and the assay plate ID (*apid*).

R Input

```
> ## List the plates and plot one
> l1data[ , head(unique(apid))]
```

R Output

```
[1] "AP0001" "AP0003" "AP0002"
```

R Input

```
> tcplPlotPlate(dat = l1data, apid = "AP0002")
```

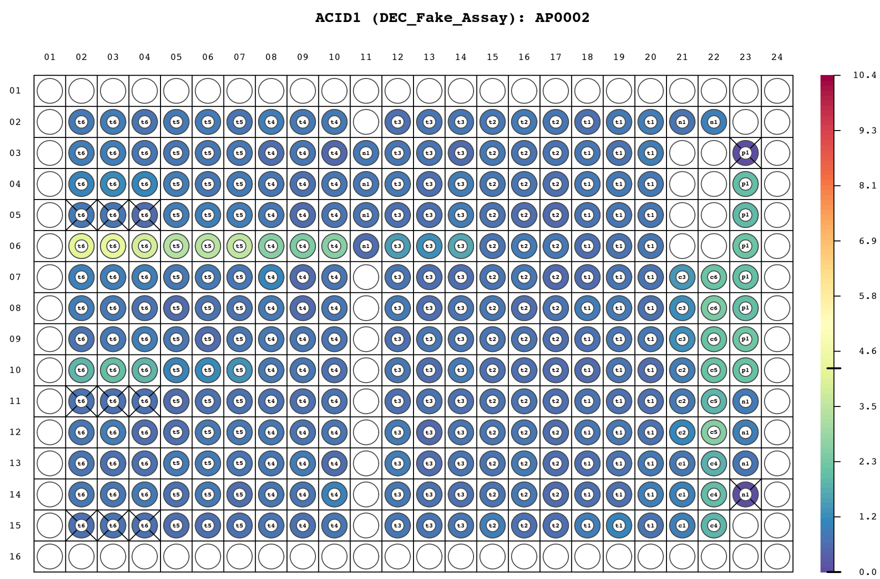

Figure 1: An assay plate diagram. The color indicates the raw values according to the key on the left. The bold lines on the key show the distribution of values for the plate on the scale of values across the entire assay. The text inside each well shows the well type and concentration index. For example, “t4” indicates a test compound at the fourth concentration. The wells with an “X” have a well quality of 0.

Level 2

Level 2 processing removes data where the well quality equals 0, and defines the corrected value (*cval*) field. The purpose of this step is to allow for any manipulation or correction of the raw values at the assay component level. Examples of correction methods could range from basic transformations like a logarithm, to complex transformations like a spacial noise reduction algorithm. Currently the `tcpl` package only consists of basic transformations, but could be expanded in future releases. Level 2 correction does not include normalization methods – normalization should occur at level 3.

To promote reproducibility, all method assignments must occur through the database. In other words, the user cannot simply pass an argument to the processing function and apply a method to the data. In general, the available methods are stored in the "l#_methods" tables, where "#" corresponds to the level. The corresponding "l#_id" tables store the method assignments, along with the method execution order. The processing functions load the methods for the given assay component (or assay endpoint in subsequent levels) using the `tcplLoadMthd()` function:

R Input

```
> tcplLoadMthd(lvl = 2L, id = 1L)
```

R Output

```
   acid mthd mthd_id ordr
1:    1 none       1    1
```

In the example data the only method assigned to acid 1 is "none". Every assay component needs at least one correction method assigned to complete level 2 processing. The method can be "none", as in the example dataset. Suppose the user wants to add a correction method. First list out the methods with `tcplListMthd`, then use `tcplAssignMthd` to assign the method(s) to the assay component after the old methods are cleared using `tcplClearMthd`.

R Input

```
> head(tcplListMthd(lvl = 2L))
```

R Output

```
   l2_mthd_id l2_mthd
1:          1    none
2:          2    log2
3:          3   rmneg
4:          4  rmzero
5:          5  mult25
6:          7 mult100
                                               desc
1:                          apply no level 2 method
2:                                log2 all raw data
3: remove negative values prior to logging values
4:        remove 0 values prior to logging values
5:                             multiply values by 25
6:                            multiply values by 100
```

R Input

```
> tcplClearMthd(lvl = 2L, id = 1L)
```

R Output

```
Completed delete cascade for 1 ids (0.06 secs)
[1] TRUE
```

R Input

```
> tcplAssignMthd(lvl = 2L, id = 1L, mthd_id = c(4L, 3L),
                 ordr = 1:2)
```

R Output

```
Completed delete cascade for 1 ids (0.06 secs)
```

R Input

```
> tcplLoadMthd(lvl = 2L, id = 1L)
```

R Output

```
   acid   mthd mthd_id ordr
1:    1 rmzero       4    1
2:    1  rmneg       3    2
```

In the example above, the old methods were cleared, then reassigned. The final result shows the new set of correction methods assigned to the data: in order, remove zero values then remove negative values. Also notice changing the methods (clearing or assigning), triggers a delete cascade similar to the processing. Every time methods are changed all data and subsequent data for the level affected gets deleted. The delete cascade ensures the methods always correctly correspond to the data in the database. With the methods assigned and checked we can carryon with level 2 processing:

R Input

```
> ## Do level 2 processing for acid 1
> tcpl2(ac = 1L)
```

R Output

```
Loaded L1 ACID1 (864 rows; 0.02 secs)
Processed L2 ACID1 (847 rows; 0.02 secs)
Completed delete cascade for 1 ids (0.06 secs)
Wrote L2 ACID1 (847 rows; 0.08 secs)
[1] TRUE
```

For the complete list of level 2 correction methods currently available, see `tcplListMthd(lvl = 2L)`, or `?L2_Methods` for more detail. The coding methodly used to impliment the methods is beyond the scope of this vignette, but in brief, the method names in the databases correspond to a function name in the list of functions returned by `tcplCorrFuncs()` (the `tcplCorrFuncs()` function is not exported, and not intended for use by the user). Each of the functions in the list given by `tcplCorrFuncs()` only return expression objects that `tcpl2` executes in the local function environment to avoid making additional copies of the data in memory.

Level 3

Level 3 processing converts the assay component to assay endpoint(s) and defines the normalized response value field (*resp*), logarithm concentration field (*logc*), and optionally, the baseline value (*bval*) and positive control value (*pval*) fields. The purpose of level 3 is to normalize the corrected values to either the percentage of a control, or to fold-change from baseline. The processing aspect of level 3 is almost completely analagous to level 2, except the user has to be careful about using assay component versus assay endpoint.

As discussed in the Assay Structure section (page 4), an assay component can have more than one assay endpoint allowing for multiple normalization approaches. Mulitple normalization approaches become necessary when the assay component detects both gain and loss signals. As discussed in the following section, the curvefitting algorithm only fits in the positive direction, so negative direction curves must be transformed to the positive direction during normalization.

The user first needs to check which assay endpoints stem from the the assay component queued for processing. It is also important to check the normalization methods assigned to the assay endpoints.

R Input

```
> ## Look at the assay endpoints for acid 1
> tcplLoadAeid(fld = "acid", val = 1L)
```

R Output

```
   acid aeid               aenm
1:    1    1 DEC_Fake_Assay_up
```

R Input

```
> ## Look at the normalization methods assigned to aeid 1
> tcplLoadMthd(lvl = 3L, id = 1L)
```

R Output

```
   aeid                        mthd mthd_id ordr
1:    1     bval.apid.lowconc.med       2    1
```

```
2:    1 pval.apid.medpcbyconc.max      3    2
3:    1                   resp.pc      5    3
```

The example dataset only includes one assay endpoint, "DEC_Fake_Assay_up". The assay endpoint has three methods assigned, ending with "resp.pc". When assay endpoints require normalization it is important to consider the necessary elements. The pipeline only accepts zero-centered data, so all response values must eventually be in percent of control or logarithm fold-change units. When normalizing to a control, an assay endpoint needs at least three normalization methods: one to define the baseline value, one to define the control value, and the method to calculate percent of control ("resp.pc"). Normalizing to fold-change also requires at least three methods: one to define the baseline value, one to calculate the fold change, and one take a logarithm. The methods for defining a baseline value have the "bval" prefix, and the methods for defining the control value have the "pval" prefix. The formluae for calculating the the percent of control and fold-change response values are listed in equations 1 and 2, respectively.

The percent of control and fold-change values, respectively, as:

$$resp = \frac{cval - bval}{pval - bval}100 \tag{1}$$

$$resp = cval/bval \tag{2}$$

For a complete list of normalization methods see `?L3_Methods`. With the assay endpoints and normalization methods defined, the user can proceed with level 3 processing fo the assay component.

R Input

```
> ## Do level 3 processing for acid 1
> tcpl3(ac = 1)
```

R Output

```
Loaded L2 ACID1 (847 rows; 0.02 secs)
Processed L3 ACID1 (AEIDS: 1; 847 rows; 0.11 secs)
Completed delete cascade for 1 ids (0.01 secs)
Wrote L3 ACID1 (AEIDS: 1; 847 rows; 0.04 secs)
[1] TRUE
```

Notice the `tcpl3` function takes an assay component id, NOT an assay endpoint id. The user must assign normalization methods to assay endpoint, then do the processing by assay component. The `tcpl3` function will process all endpoints in the database for a given component. If one endpoint does not have appropriate methods assigned, and fails, the processing for the entire component fails.

Level 4

Level 4 processing splits the data into concentration series by sample and assay endpoint, then models the activity of each concentration series. Activity is only modeled in the positive direction. More information on readouts with both directions is available in the previous section.

Only concentration series with evidence of activity go through the fitting algorithm. The first step in level 4 processing is to remove the well types with only one concentration. To establish the noiseband for the assay endpoint, the baseline median absolute deviation ($bmad$) is calculated as the median absolute deviation of the response values for test compounds where the concentration index equals 1 or 2. The calculation to define $bmad$ is done once across the entire assay endpoint. If additional data is added to the database for an assay component, the bmad values for all associated assay endpoints will change.

Before the model parameters are estimated, the `tcpl4` function calculates a set of summary values for each concentration series: the minimum and maximum response, minimum and maximum log concentration, the number of concentrations, points, and replicates, the maximum mean and median with the concentration at which they occur, and the number of medians greater than $3bmad$. When referring to the concentration series the "mean" and "median" values are defined as the mean or median of the response values at every concentration. In other words, the maximum median is the maximum of all median values across the concentration series.

Concentration series have to have at least four concentrations and at least one median value to enter the fitting algorithm. All models draw from the Student's t-distribution with 4 degrees of freedom. The wider tails in the t-distribution diminish the influence of outlier values, and produce more robust estimates than the more commonly used normal distribution. The robust fitting removes the need for any outlier elimination before fitting. The fitting algorithm utilizes maxmimum likelihood estimates parameters for three models as defined below in equations 3 to 16.

Let $t(z,\nu)$ be the Student's t-ditribution with ν degrees of freedom, y_i be the observed response at the i^{th} observation, and μ_i be the estimated response at the i^{th} observation. We calculate z_i as:

$$z_i = \frac{y_i - \mu_i}{\exp(\sigma)} \tag{3}$$

where σ is the scale term. Then the log-likelyhood is:

$$\sum_{i=1}^{n}[\ln(t(z_i, 4)) - \sigma] \tag{4}$$

where n is the number of observations.

The first model fit in the fitting algorithm is a constant model at 0, abbreviated "cnst". The constant model only has one paramter, the scale term. For the constant model μ_i is given by:

$$\mu_i = 0. \tag{5}$$

The second model in the fitting algorithm is a constrained Hill model (hill) where the bottom asymptote is forced to 0. Including the scale parameter the Hill model has four parameters. Let tp be the top asymptote, ga be the AC50[2] in the gain direction, gw be the Hill coefficient in the gain direction, and x_i be the log concentration at the i^{th} observation. Then μ_i for the Hill model is given by:

$$\mu_i = \frac{tp}{1 + 10^{(ga - x_i)gw}} \tag{6}$$

with the constraints

$$0 \leq tp \leq 1.2 \text{max resp}, \tag{7}$$

$$\text{min logc} - 2 \leq ga \leq \text{max logc} + 0.5, \tag{8}$$

and

$$0.3 \leq gw \leq 8. \tag{9}$$

The third model in the fitting alrogirthm is a constrained gain-loss model (gnls), defined as a product of two Hill models with a shared top asymptote and both bottom asymptotes equal to 0. Including the scale term, the gain-loss model has six parameters. Let tp be the shared top asymptote, ga be the AC50 in the gain direction, gw be the Hill coefficient in the gain direction, la be the AC50 in the loss direction, lw be the Hill coefficient in the loss direction, and x_i be the log concentration at the i^{th} observation. Then μ_i for the gain-loss model is given by:

$$\mu_i = tp \left(\frac{1}{1 + 10^{(ga - x_i)gw}} \right) \left(\frac{1}{1 + 10^{(x_i - la)lw}} \right) \tag{10}$$

with the constraints

$$0 \leq tp \leq 1.2 \text{max resp}, \tag{11}$$

$$\text{min logc} - 2 \leq ga \leq \text{max logc}, \tag{12}$$

$$0.3 \leq gw \leq 8, \tag{13}$$

$$\text{min logc} - 2 \leq la \leq \text{max logc} + 2, \tag{14}$$

$$0.3 \leq lw \leq 18, \tag{15}$$

and

$$ga - la > 0.25. \tag{16}$$

Level 4 does not utilize any assay endpoint-specific methods; the user only needs to run the `tcpl4` function. **Level 4 processing and all subsequent processing is done by assay endpoint, NOT assay component**. The previous section showed how to find the assay endpoints for an assay component using the `tcplLoadAeid` function. The example dataset includes one assay endpoint with the name "DEC_Fake_Assay_up" and ID 1.

[2]The AC50 is the activity concentration at 50%, or the concentration where the modeled activity equals 50% of the top asymptote.

R Input

```
> ## Do level 4 processing for aeid 1 and load the data
> tcpl4(ae = 1)
```

R Output

```
Loaded L3 AEID1 (795 rows; 0.02 secs)
Processed L4 AEID1 (795 rows; 1.03 secs)
Completed delete cascade for 1 ids (0.01 secs)
Wrote L4 AEID1 (795 rows; 0.11 secs)
[1] TRUE
```

R Input

```
> l4data <- tcplLoadData(lvl = 4L, fld = "aeid", val = 1L)
> dim(l4data)
```

R Output

```
[1] 42 51
```

The level 4 data includes 52 variables, including the ID fields. A complete list of level 4 fields is avialable in Appendix A. The example dataset contains 42 concentration series. The level 4 data includes the fields *cnst*, *hill*, and *gnls* indicating the convergance of the model where a value of 1 means the model converged and a value of 0 means the model did not converge. N/A values indiciate the fitting algorithm did not attempt to fit the model. *cnst* will be N/A when the concentration series had less than 4 concentrations; *hill* and *gnls* will be N/A when none of the medians were greater than or equal to $3bmad$. Similarly, the *hcov* and *gcov* fields indicate the success in inverting the Hessian matrix. Any NaN values in the parameter standard deviation fields indicate the Hessian matrix was not positive definite, and the standard deviation values lose meaning. In Figure 2 the *hill* field is used to find potentially active compounds to visualize with the `tcplPlotL4ID` function.

The model summary values in Figure 2 include Akaike Information Criterion (AIC), probability, and the root mean square error (RMSE). Let $\log(\mathcal{L}(\hat{\theta}, y))$ be the log-likelihood of the model $\hat{\theta}$ given the observed values y, and K be the number of parameters in $\hat{\theta}$, then,

$$\text{AIC} = -2\log(\mathcal{L}(\hat{\theta}, y)) + 2K \tag{17}$$

The probability, ω_i, is defined as the weight of evidence that model i is the best model, given that one of the models must be the best model. Let Δ_i be the difference $\text{AIC}_i - \text{AIC}_{min}$ for the i^{th} model. If R is the set of models, then ω_i is given by:

$$\omega_i = \frac{\exp\left(-\frac{1}{2}\Delta_i\right)}{\sum_{i=1}^{R} \exp\left(-\frac{1}{2}\Delta_r\right)} \tag{18}$$

The RMSE is given by:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \mu_i)^2}{N}} \tag{19}$$

where N is the number of observations, and μ_i and y_i are the estimated and observed values at the i^{th} observation, respectively.

R Input

```
> ## List the l4ids where the Hill model converged and plot 1
> l4data[hill == 1, l4id]
```

R Output

```
 [1]  1  3  6 16 18 19 26 33 35 38 39 40 42
```

R Input

```
> tcplPlotL4ID(l4id = 35, lvl = 4L)
```

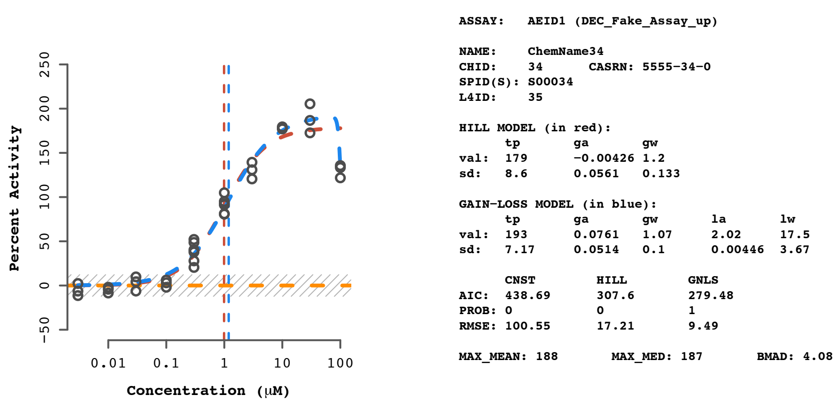

Figure 2: An example level 4 plot for a single concentration series. The orange dashed line shows the constant model, the red dashed line shows the Hill model, and the blue dashed line shows the gain-loss model. The gray striped box shows the baseline region, $0 \pm 3bmad$. The summary panel shows assay endpoint and sample information, the parameters and standard deviations for the Hill and gain-loss models, and model summary values.

Level 5

Level 5 processing determines the winning model and activity for the concentration series, bins all of the concentration series into categories, and calculates additional point of departure estimates based on the activity cutoff.

The model with lowest AIC value is selected as the winning model (*modl*), and is used to determine the activity or hit-call for the concentration series. If two models have equal AIC values, the simpler model (the model with less parameters) wins the tie. All of the parameters for the winning model are stored at level 5 with the prefix "modl_" to fascilitate easier queries. For a concentration series to get an active hit-call either the Hill or gain-loss must be selected. In addition to selecting the Hill or gain-loss model, the modeled and observed response must meet an efficacy cutoff.

The final cutoff value (*coff*) is defined as the maximum of three values: 20% change, $3bmad$, and an additional cutoff. The additional cutoff is given by the level 5 methods, analagous to the correction and normalization methods used in levels 2 and 3. Currently, the additional cutoff methods consist of additional scalars of bmad.

R Input

```
> ## Check aeid 1 for level 5 methods
> tcplLoadMthd(lvl = 5L, id = 1L)
```

R Output

```
   aeid  mthd mthd_id
1:    1 bmad5       2
```

For the example data the additional cutoff value is $5bmad$. The final cutoff value is stored at level 5. The *bmad* for the example data is 4.08, making the final cutoff will be $5bmad$, or 20.41, because $5bmad$ is by definition greater than $3bmad$ and, in this example, greater than 20. If the Hill or gain-loss model wins, and the estimated top paramter for the winning model (*modl_tp*) and the maximum median value (*max_med*) are both greater than or equal to the cutoff (*coff*), the concentration series is considered active and the hit-call (*hitc*) is set to 1.

The hit-call can be 1, 0, or -1. A hit-call of 1 or 0 indicates the concentration series is active or inactive, respectively, according to the analysis; a hit-call of -1 indicates the concentration series had less than 4 concentrations.

When applicable, the activity concentration at baseline (ACB or *modl_acb*) and the activity concentration at cutoff are calculated for the winning model (ACC or *modl_acc*). The ACB and ACC are defined as the cocentration where the estimated model value equals $3bmad$ and the cutoff, respectively.

All concentration series fall into a single fit category (*fitc*), defined by the leaves on the tree structure in Figure 3. Concentration series in the same category will have similar characteristics, and often look very similar. Categorizing all of the series allows for faster quality control checking, and fast identification of potential false results. The first split differentiates series by hit-call. Series with a hit-call of -1 go into fit category 2. The following two paragraphs will outline the logic for the active and inactive branches.

The first split in the active branch differentiates series by the model winner, Hill or gain-loss. For each model, the next split is defined by the efficacy of it's top parameter in relation to the cutoff. The top values is either less than $1.2\mathit{coff}$ or greater than or equal to $1.2\mathit{coff}$. Finally, series on the active branch go into leaves based on the position of the AC50 parameter in relation to the tested concentration range. For comparison purposes the activity concentration at 95% (AC95) is calculated, but not stored[3]. Series with AC50 values less than the minimum concentration tested (*logc_min*) go into the "<=" leaves, series with AC50 values greater than the minimum tested concentration and AC95 values less than maximum tested concentration (*logc_max*) go into the "==" leaves, and series with AC95 values greater than the maximum concentration tested go into the ">=" leaves.

The inactive branch is first divided by whether any median values were greater than or equal to $3bmad$. Series with no evidence of activity go into fit cateogry 4. Similar to the active branch, series with evidence for activity are separated by the model winner. The Hill and gain-loss portions of the inactive branch follow the same logic. First series diverge by the efficacy of their top paramter in relation to the cutoff: $\mathit{modl\_tp} < 0.8\,\mathit{coff}$ or $\mathit{modl\_tp} \geq 0.8\,\mathit{coff}$. Then the same comparison is made on the top values of the losing model. If the losing model did not converge, the series go into the "DNC" category. If the losing model top value is greater than or equal to $0.8\mathit{coff}$, the series are split based on whether the losing model top surpassed the cutoff. On the constant model branch, if neither top parameter is greater than or equal to $0.8bmad$, the series goes into fit category 7. If one of the top parameters is greater than or equal to $0.8\mathit{coff}$, the series goes into fit cateory 9 or 10 based on whether one of the top values surpasssed the cutoff.

With the level 5 method assigned, the user can carryout level 5 processing:

R Input

```
> ## Do level 5 processing for aeid 1 and load the data
> tcpl5(ae = 1L)
```

R Output

```
Loaded L4 AEID1 (42 rows; 0.01 secs)
Processed L5 AEID1 (42 rows; 0.05 secs)
Completed delete cascade for 1 ids (0.01 secs)
Wrote L5 AEID1 (42 rows; 0.02 secs)
[1] TRUE
```

[3] Any activity concentration value or estimated model values for a given concentration can be calculated using the `tcplACXX` and `tcplACVal` functions, respectively

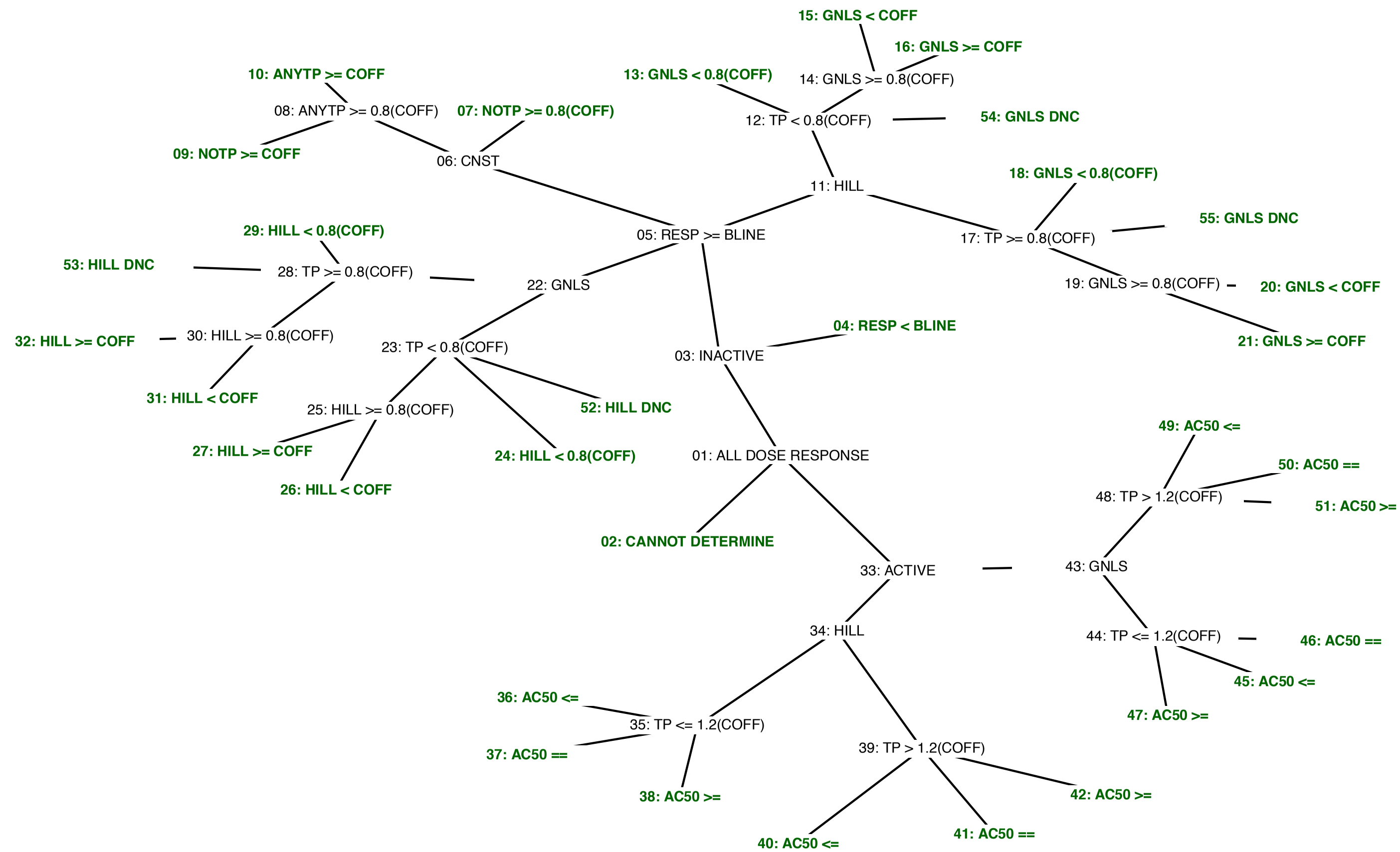

Figure 3: The categories used to bin each fit. Each fit falls into one leaf of the tree. The leaves are indicated by bold green font.

R Input

```
> l5data <- tcplLoadData(lvl = 5L, fld = "aeid", val = 1L)
```

R Input

```
> tcplPlotL4ID(l4id = 33L, lvl = 5L)
```

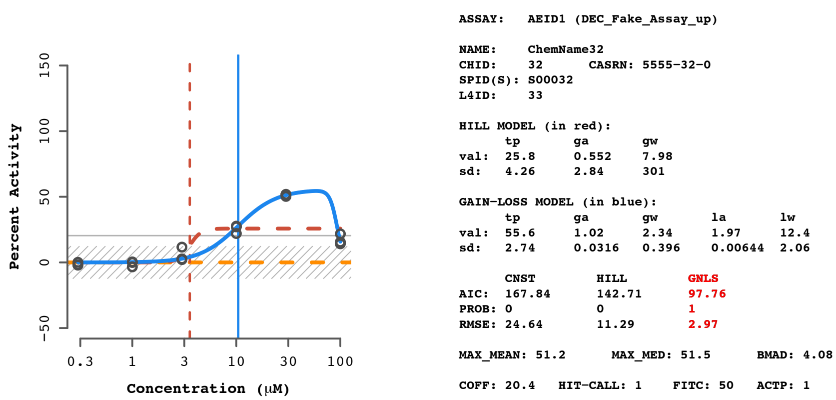

Figure 4: An example level 5 plot for a single concentration series. The solid line and model highlighting indicate the model winner. The horizontal line shows the cutoff value. The summary values include the cutoff, hit-call, fit category and activity probability.

Figure 4 shows an example of a concentration series in fit category 50, indicating the series is active with and the gain-loss model won with a top value greater than $1.2\mathit{coff}$ and an AC50 value within the tested concentration range. The `tcplPlotFitc` functions shows the distribution of concentration series across the fit category tree (Figure 5).

The distribution in Figure 5 shows at least 1 concentration series fell into fit category 21. Following the logic discussed previously, fit cateogry 21 indicates an inactive series where the Hill model was selected, the top asymptote for the Hill model was greater than $0.8\mathit{coff}$ and the gain-loss top asymptote was greater than or equal to the cutoff. The series in fit category 21 can be found easily in the level 5 data:

R Input

```
> tcplPlotFitc(fitc = l5data$fitc)
```

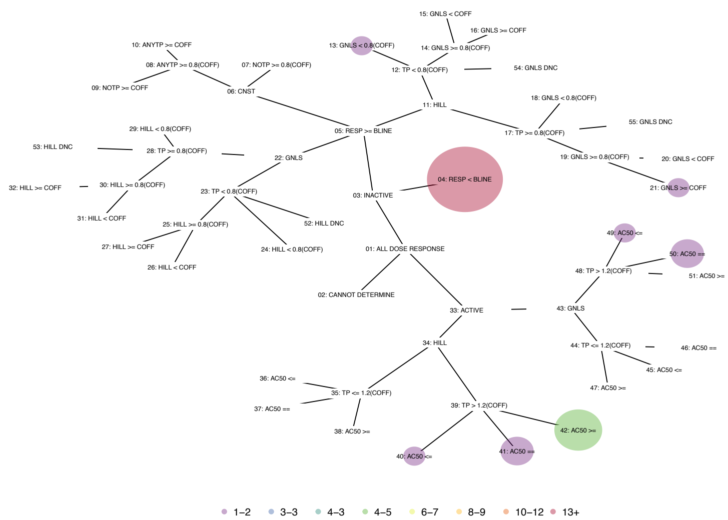

Figure 5: The distribution of concentration series by fit category for the example data. Both the size and color of the circles indicate the number of concentration series. The legend gives the range for number of concentration series by color.

R Input

```
> l5data[fitc == 21,
         list(l4id, hill_tp, gnls_tp, max_med, coff, hitc)]
```

R Output

```
   l4id  hill_tp  gnls_tp  max_med    coff hitc
1:   18 22.16028 22.16028 13.34993 20.4147    0
```

The output and plot in Figure 6 show the series (l4id 18) in fit cateogry 21. The *hill_tp* and *gnls_tp* parameters are equal and greater than *coff*, however, the maximum median value ($\max_m ed$) is not greater than the cutoff making the series inactive.

Level 6

Level 6 processing applies methods to flag concentration series to help identify potential false positive and false negative hit-calls or explain apparent anomalies

R Input
```
> tcplPlotL4ID(l4id = 18L, lvl = 5L)
```

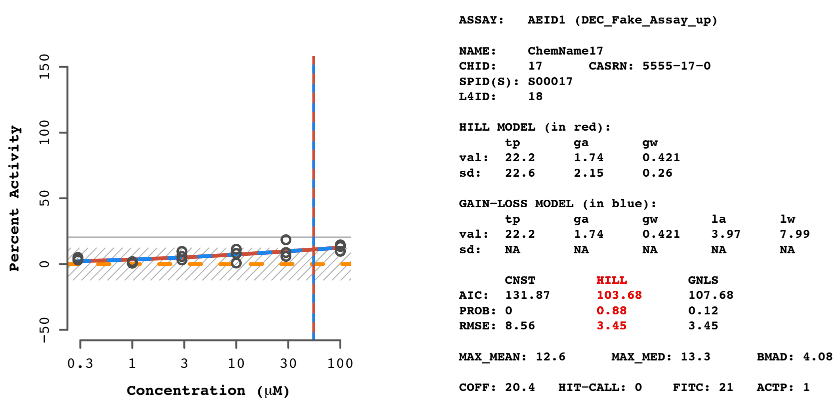

Figure 6: Level 5 plot for l4id 18 showing the series in fit category 21.

in the data. Each flag has is defined by a level 6 method that has to be assigned to each assay endpoint. Unlike previous levels, an assay endpoint does not need any level 6 methods assigned to complete level 6 processing. Analagous to earlier levels, the assigned level 6 methods can be loaded with the `tcplLoadMthd` function:

R Input
```
> tcplLoadMthd(lvl = 6L, id = 1L)
```

R Output
```
   aeid              mthd mthd_id nddr
1:    1 singlept.hit.high       6    0
2:    1  singlept.hit.mid       7    0
3:    1    multipoint.neg       8    0
4:    1             noise      10    0
5:    1        border.hit      11    0
6:    1       border.miss      12    0
7:    1      rep.mismatch      14    1
8:    1      gnls.lowconc      15    0
```

The example data has 8 level 6 methods assigned to aeid 1. The *nddr* field in the output from `tcplLoadMthd` for level 6 indicates if the method requires data for the individual points. Methods with a value of 0 in the *nddr* field only require the modeled/summary information at levels 4 and 5. If any method requires the individual point data the `tcpl6` has to load additional data and the processing time increases.

R Input

```
> ## Do level 6 processing for aeid 1 and load the data
> tcpl6(ae = 1L)
```

R Output

```
Loaded L5 AEID1 (42 rows; 0.03 secs)
Processed L6 AEID1 (42 rows; 0.26 secs)
Completed delete cascade for 1 ids (0 secs)
Wrote L6 AEID1 (3 rows; 0.02 secs)
[1] TRUE
```

R Input

```
> l6data <- tcplLoadData(lvl = 6L, fld = "aeid", val = 1L)
> l6data
```

R Output

```
   aeid l6id l4id l5id   spid l6_mthd_id
1:    1    1   19   19 S00018          6
2:    1    2   40   40 S00039         10
3:    1    3   18   18 S00017         12
                                     flag fval fval_unit
1: Only highest conc above baseline, active   NA        NA
2:                              Noisy data   NA        NA
3:                     Borderline inactive   NA        NA
```

In the example data 3 out of the 42 concentration series were flagged in the level 6 processing. Series not flagged in the level 6 processing do not get stored at level 6. Each series-flag combination is a separate entry in the level 6 data. Or in other words, if a series has multiple flags it will show up on multiple rows in the output. In the example data no series has more than one flag. The first series listed in level 6 has the flag "Only highest conc above baseline, active," meaning the only median value in the series greater than $3bmad$ was at the highest tested concentration. The plot for l4id 19 (Figure 7) illustrates the only activity occuring at the highest concentration tested. The series certainly has statistical evidence of activity, but the flagging allows users to quickly identify potential problems and potentially differentiate the series in Figure 7 from a series similar to Figure 2 where both asymptotes are resolved and mutliple concentrations fell outside the noiseband.

R Input

```
> tcplPlotL4ID(l4id = 19L, lvl = 6L)
```

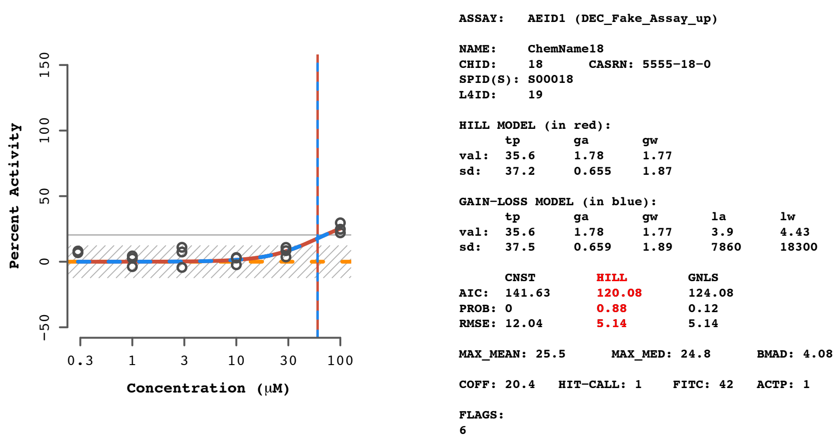

Figure 7: An example level 6 plot for a single concentration series. All level 6 method ID (*l6_mthd_id*) values concatenated in the flags secion. If flags have an associated value (*fval*), the value will be shown in parentheses to the right of the level 6 method ID.

Single Concentration Screening

The single concentration screening aspect of the package is still under development. The current single screen functions included in the package are not intended for use.

A Field Explaination/Database Stucture

Table 2: List of data tables in the `tcpl` databases.

Table Name	Description
agg_level4	A look-up table between level3 and level4 storing the level 0 through level 4 primary keys
assay_component_map	Assay component source names and their corresponding assay component ids†
l2_acid	Level 2 method assignments at the assay component level
l2_methods	Level 2 methods
l3_aeid	Level 3 method assignments at the assay endpoint level
l3_methods	Level 3 methods
l5_aeid	Level 5 method assignemnts at the assay endpoint level
l5_fit_categories	The level 5 fit categories
l5_methods	Level 5 methods
l6_aeid	Level 6 method assignments at the assay endpoint level
l6_methods	Level 6 methods
level0	Level 0 multiple concentration
level1	Level 1 multiple concentration
level2	Level 2 multiple concentration
level3	Level 3 multiple concentration
level4	Level 4 multiple concentration
level5	Level 5 multiple concentration
level6	Level 6 multiple concentration
single0	Level 0 single concentration
single1	Level 1 single concentration
single2	Level 2 single concentration

†Assay component source name (*acsn*) used in level 0 processing to write data to `tcpl` databases

Table 3: Fields in agg_level4 table.

Field	Description
aeid	Assay endpoint ID
l0id	Level 0 ID
l1id	Level 1 ID
l2id	Level 2 ID
l3id	Level 3 ID
l4id	Level 4 ID

Table 4: Fields in assay_component_map table.

Field	Description
acid	Assay component ID
acsn	Assay component source name

Table 5: Fields in l2_acid table.

Field	Description
l2_mthd_id	Level 2 method ID assigned to the assay component
acid	Assay component ID for the assigned method
exec_ordr	Execute order for the method
created_date	Date created
modified_date	Date modified
modified_by	User that created/modified the assignment

Table 6: Fields in l2_methods table.

Field	Description
l2_mthd_id	Level 2 method ID
l2_mthd	Level 2 method name
desc	Level 2 method description

Date and user fields not listed to save space, see Table 5.

Table 7: Fields in l3_aeid table.

Field	Description
l3_mthd_id	Level 3 method ID assigned to the assay endpoint
aeid	Assay endpoint ID for the assigned method
exec_ordr	Execute order for the method

Date and user fields not listed to save space, see Table 5.

Table 8: Fields in l3_methods table.

Field	Description
l3_mthd_id	Level 3 method ID
l3_mthd	Level 3 method name
desc	Level 3 method description

Date and user fields not listed to save space, see Table 5.

Table 9: Fields in l5_aeid table.

Field	Description
aeid	Assay endpoint ID for the assigned method
l5_mthd_id	Level 5 method ID assigned to the assay endpoint

Date and user fields not listed to save space, see Table 5.

Table 10: Fields in l5_fit_categories table.

Field	Description
fitc	Fit category
parent_fitc	Parent fit category
name	Fit category name
xloc	x-axis location for plotting purposes
yloc	y-axis location for plotting purposes

Table 11: Fields in l5_methods table.

Field	Description
l5_mthd_id	Level 5 method ID
l5_mthd	Level 5 method name
desc	Level 5 method description

Date and user fields not listed to save space, see Table 5.

Table 12: Fields in l6_aeid table.

Field	Description
aeid	Assay endpoint ID for the assigned method
l6_mthd_id	Level 6 method ID assigned to the assay endpoint

Date and user fields not listed to save space, see Table 5.

Table 13: Fields in l6_methods table.

Field	Description
l6_mthd_id	Level 6 method ID
l6_mthd	Level 6 method name
desc	Level 6 method descripion
nddr	1 if the method requires individual point data, else 0

Date and user fields not listed to save space, see Table 5.

Table 14: Fields in level0 table.

Field	Description
l0id	Level 0 ID
acid	Assay component id
spid	Sample ID
cpid	Chemical plate ID
apid	Assay plate ID
rowi	Assay plate row index
coli	Assay plate column index
wllt	Well type†
wllq	1 if the well quality was good, else 0‡
conc	Concentration in mircomolar
rval	Raw assay component value/readout from vendor
srcf	Filename of the source file containing the data

Date and user fields not listed to save space, see Table 5.
†Information about well types available in Appendix B
‡Entries with well quality values of 0 get removed during processing

Table 15: Fields in level1 table.

Field	Description
l1id	Level 1 ID
l0id	Level 0 ID
acid	Assay component ID
cndx	Concentration index
repi	Replicate index

Date and user fields not listed to save space, see Table 5.

Table 16: Fields in level2 table.

Field	Description
l2id	Level 2 ID
l0id	Level 0 ID
acid	Assay component ID
l1id	Level 1 ID
cval	Corrected value

Date and user fields not listed to save space, see Table 5.

Table 17: Fields in level3 table.

Field	Description
l3id	Level 3 ID
aeid	Assay endpoint ID
l0id	Level 0 ID
acid	Assay component ID
l1id	Level 1 ID
l2id	Level 2 ID
bval	Baseline value
pval	Positive control value
logc	Log base 10 concentration
resp	Normalized response value

Date and user fields not listed to save space, see Table 5.

Table 18: Fields in level4 table (Part 1).

Field	Description
l4id	Level 4 ID
aeid	Assay endpoint ID
spid	Sample ID
bmad	Baseline median absolute deviation
resp_max	Maximum response value
resp_min	Minimum response value
max_mean	Maximum mean response value
max_mean_conc	Log concentration at *max_mean*
max_med	Maximum median response value
max_med_conc	Log concentration at *max_med*
logc_max	Maximum log concentration tested
logc_min	Minimum log concentration tested
cnst	1 if the constant model converged, 0 if failed to convere, N/A if series had less than four concentrations
hill	1 if the hill model converged, 0 if failed to converge, N/A if series had less than four concentrations or if $max\_med < 3bmad$
hcov	1 if the hill model Hessian matrix could be inverted, else 0
gnls	1 if the gain-loss model converged, 0 if failed to converge, N/A if series had less than four concentrations or if $max\_med < 3bmad$
gcov	1 if the gain-loss model Hessian matrix could be inverted, else 0
cnst_er	Scale term for the constant model
cnst_aic	AIC for the constant model
cnst_rmse	RMSE for the constant model
cnst_prob	Probability the constant model is the true model
hill_tp	Top asymptote for the Hill model
hill_tp_sd	Standard deviation for *hill_tp*
hill_ga	AC50 for the Hill model
hill_ga_sd	Standard deviation for *hill_ga*

Table 19: Fields in level4 table (Part 2).

Field	Description
hill_gw	Hill coefficient
hill_gw_sd	Standard deviation for *hill_gw*
hill_er	Scale term for the Hill model
hill_er_sd	Standard deviation for *hill_er*
hill_aic	AIC for the Hill model
hill_rmse	RMSE for the Hill model
hill_prob	Probability the Hill model is the true model
gnls_tp	Top asymptote for the gain-loss model
gnls_tp_sd	Standard deviation for *gnls_tp*
gnls_ga	AC50 in the gain direction for the gain-loss model
gnls_ga_sd	Standard deviation for *gnls_ga*
gnls_gw	Hill coefficient in the gain direction
gnls_gw_sd	Standard deviation for *gnls_gw*
gnls_la	AC50 in the loss direction for the gain-loss model
gnls_la_sd	Standard deviation for *gnls_la*
gnls_lw	Hill coefficient in the loss direction
gnls_lw_sd	Standard deviation for *gnls_lw*
gnls_er	Scale term for the gain-loss model
gnls_er_sd	Standard deviation for *gnls_er*
gnls_aic	AIC for the gain-loss model
gnls_rmse	RMSE for the gain-loss model
gnls_prob	Probability the gain-loss model is the true model
nconc	Number of concentrations tested
npts	Number of points in the concentration series
nrep	Number of replicates in the concentration series
nmed_gtbl	Number of median values greater than $3bmad$
tmpi	Ignore, temporary index used for uploading purposes

Date and user fields not listed to save space, see Table 5.

Table 20: Fields in level5 table.

Field	Description
l5id	Level 5 ID
l4id	Level 4 ID
aeid	Assay endpoint ID
modl	Winning model: “cnst”, “hill”, or “gnls”
hitc	Hit-call, 1 if active, 0 if inactive
fitc	Fit category†
coff	Final cutoff value
actp	Activity probability ($1 - cnst\_prob$)
modl_er	Scale term for the winning model
modl_tp	Top asymptote for the winning model
modl_ga	Gain AC50 for the winning model
modl_gw	Gain Hill coefficient for the winning model
modl_la	Loss AC50 for the winning model
modl_lw	Loss Hill coefficient for the winning model
modl_prob	Probability for the winning model
modl_rmse	RMSE for the winning model
modl_acc	Acitivty concentration at cutoff for the winning model
modl_acb	Acitivty concentration at baseline for the winning model
modl_ac10	AC10 for the winning model

Date and user fields not listed to save space, see Table 5.
† Fit category schema in Figure 3

Table 21: Fields in level6 table.

Field	Description
l6id	Level 6 ID
l5id	Level 5 ID
l4id	Level 4 ID
aeid	Assay endpoint ID
l6_mthd_id	Level 6 method ID
flag	Text text output for the level 6 method
fval	Value from the flag method, if applicable
fval_unit	Units for *fval*, if applicable

Date and user fields not listed to save space, see Table 5.

Table 22: List of assay annotation tables in the `tcpl` databases.

Table Name	Description
assay	Assay-level annotation
assay_component	Assay component-level annotation
assay_component_endpoint	Assay endpoint-level annotation
assay_reagent	Assay reagent information
assay_reference	Citations for assay annotation
assay_source	Assay source-level annotation
gene	Gene identifiers and descriptions
intended_target	The intended assay target at the assay endpoint level
organism	Organism identifiers and descriptions
technological_target	The technological assay target at the assay component level

(omplete definitions of annotation fields available at `<http://epa.gov/ncct/toxcast/data.html>`.

Table 23: List of sample information tables in the `tcpl` databases.

Table Name	Description
casrn	Chemical identifiers
gsid_chemical_set	Chemical library information
sample	Sample to chemical ID map

Sample information tables stored in a seperate database internally at the ToxCast program. Likewise, queries to these tables go through the `TCPL_CHEM` database setting. For easier external use the tables have been copied into the data database. Not all fields used in processing.

Table 24: Fields in casrn table.

Field	Description
c_casrn_id	CAS registry number
c_gsid_id	DSSTox_GSID†
c_chemicalname	Chemical name

Unused fields not listed
†More information about the US EPA DSSTox program at `<http://www.epa.gov/ncct/dsstox/>`.

Table 25: Fields in sample table.

Field	Description
sa_sample_id	Sample ID, or *spid* in data tables
sa_gsid	DSSTox_GSID†

Unused fields not listed
†More information about the US EPA DSSTox program at `<http://www.epa.gov/ncct/dsstox/>`.

B Level 0 Pre-Processing

Level 0 pre-processing can be done on virtually any high-throughput/high-content screening application. In the ToxCast program, level 0 processing is done in R by vendor/dataset-specific scripts. The individual R scripts act as the "laboratory notebook" for the data, with all pre-processing decisions clearly commented and explained.

Level 0 pre-processing has to reformat the raw data into the standard format for the pipeline, and can also make manual changes to the data. All manual changes to the data should be very well documented with justification. Common examples of manual changes include fixing a sample ID typo, or changing well quality value(s) to 0 after finding obvious problems like a plate row/column missing an assay reagent.

Each row in the level 0 pre-processing data represents one well-assay component combination, containing 11 fields (Table 26). The only field in level 0 preprocessing not stored at level 0 is the assay component source name (*acsn*). The assay component source name should be some concatenation of data from the assay source file that identifies the unique assay components. When the data is loaded into the database, the assay component source name is mapped to assay component ID through the assay_component_map table in the `tcpl` databases. Assay components can have multiple assay component source names, but each assay component source name can only map to a single assay component.

The well type field is used in the processing to differentiate controls from test compounds in numerous applications, including normalization and definition of the assay noise level. Currently, the `tcpl` package includes the 8 well types in Table 27. Package users are encouraged to suggest new well types and methods to better accommodate their data.

The final step in level 0 pre-processing is loading the data into the `tcpl` databases. The `tcpl` package includes the `tcplWriteLvl0` function to load data into the databases. The `tcplWriteLvl0` function maps the assay component source name to the appropriate assay component ID, checks each field for the correct class, and checks the database for the sample IDs with well type "t". Each test compound sample ID must be included in the `tcpl` databases before loading data. The `tcplWriteLvl0` also checks each test compound for concentration values.

Table 26: Required fields in level 0 pre-processing.

Field	Description	N/A
acsn	Assay component source name	No
spid	Sample ID	No
cpid	Chemical plate ID	Yes
apid	Assay plate ID	Yes
rowi	Assay plate row index, as an integer	Yes
coli	Assay plate column index, as an integer	Yes
wllt	Well type	No
wllq	1 if the well quality was good, else 0	No
conc	Concentration in mircomolar	No†
rval	Raw assay component value/readout from vendor	Yes‡
srcf	Filename of the source file containing the data	No

The N/A column indicates whether the field can be N/A in the pre-processed data.
†Concentration can be N/A for control values only tested at a single concentration. Concentration cannot be N/A for any test compound data.
‡If the raw value is N/A, well type has to be 0.

Table 27: The possible well type values.

Well Type	Description
t	Test compound
c	Gain-of-signal control in multiple concentrations
p	Gain-of-signal control in single concentration
n	Neutral/negative control
m	Loss-of-signal control in multiple concentrations
o	Loss-of-signal control in single concentration
b	Blank well
v	Viability control

C Burst Z-Score Calculation

The `tcplVarMat` function creates chemical-by-assay matrices for the level 4 and level 5 data. When multiple sample-assay series exist for one chemical, a single series is selected by the `tcplSubsetChid` function. See `?tcplSubsetChid` for more information.

The `var` paramter for `tcplVarMat` can accept any of the level 4 or level 5 fields/variables, or one of two special variables. The first special variable, "tested", returns 0 or 1, where 1 indicates the chemical-assay pair was tested in either multiple concentration or single concentration. Chemical-assay pairs not tested in multiple concentration will be N/A in the hit-call matrix. The second special parameter, "zscore" returns a z-score based on the distribution of burst assays.

The burst assay endpoints are defined by the "burst_assay" field in the assay_component_endpoint table, where 1 indicates the assay endpoint is used in the burst distribution calculation. The example dataset only contains one assay endpoint, so a good illustrative example is beyond the scope of this vignette. However, the following code block demonstrates how the user would check which assay endpoints are used in the to define the burst distribution:

R Input
```
> tcplLoadAeid(fld = "burst_assay", val = 1)
```

R Output
```
Empty data.table (0 rows) of 3 cols: burst_assay,aeid,aenm
```

R Input
```
> ## Using val = 0 shows the example assay endpoint
> tcplLoadAeid(fld = "burst_assay", val = 0)
```

R Output
```
   burst_assay aeid              aenm
1:           0    1 DEC_Fake_Assay_up
```

For each chemical, the burst distrubtion is defined by the median and MAD of the AC50 ($modl_g a$) values for the burst endpoints where the hit-call was 1 (active). Once the burst distribution is defined for each chemical, the global burst MAD is defined as the median of all MAD values for chemicals with greater than 1 active burst endpoint. The burst median for chemicals with less two active burst endpoints is set to 3.[4] The burst z-score is calculated for each AC50 value as:

$$zscore = -\frac{modl\_ga - cyto\_pt}{global\_mad} \tag{20}$$

[4] In log base 10 micromolar units 3 is equivalent to 1 molar.

where *cyto_pt* is the burst median. All of the values to define the burst distribution are also returned by the `tcplVarMat` function when `var` is “zscore”. The burst z-score values are multiplied by -1 to make values that are more potent relative to the busrt distribution a higher positive z-score.

D Plans for V1.0 Release

- Completed documentation of all functions
- Mature single concentration screening
- Register sample/assay functions for adding samples and assays to `tcpl` databases easily from the R interface
- Update annotation function(s) for updating the assay and possibly sample annotations ??